IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION PAPERS

OF

EDWARD COLLES NEVILL

FOR

INSTRUCTION INTERPRETATION WITHIN A DATA PROCESSING SYSTEM

BACKGROUND OF THE INVENTION

Field of the Invention

This invention relates to data processing systems. More particularly, this invention relates to data processing systems that have an instruction interpreter that replaces a slow form instruction with a fast form instruction and that operates using a separate instruction store and data store.

Description of the Prior Art

It is known to provide Harvard architecture systems in which a separate data store and instruction store are provided. The separate data store and instruction store may typically be in the form of a separate data cache and instruction cache. Whilst there are advantages associated with such an arrangement, one problem it produces is how to deal with instruction code that is dynamically altered at runtime. In particular, it is known to provide an instruction interpreter that will modify a slow form of instruction to a fast form of instruction at runtime. In a Harvard system, the instructions are typically provided within a read only store and the writing of a modified form of instruction out to the data store would entail a performance reducing flush and reload of at least some portions of the data and instruction stores or risk problems due to inconsistency between different forms of the same instruction being held in the instruction store and the data store.

SUMMARY OF THE INVENTION

Viewed from one aspect the present invention provides apparatus for processing data, said apparatus comprising:

(i) a processor core;

(ii) a main memory operable to store instruction words and data words;

(iii) a data store operable to store words from said main memory accessed by a data store port of said processor core;

(iv) an instruction store operable to store words from said main memory accessed by an instruction store port of said processor core; and

(v) an instruction interpreter operable to read instruction words from said instruction store; wherein

(vi) said instruction interpreter is operable to modify a slow form instruction within said instruction store to a fast form instruction of one or more possible fast form instructions and to write said fast form instruction to said data store, said slow form instruction and said fast form instruction having a common functionality when executed by said interpreter; and

(vii) said instruction interpreter is operable upon reading a slow form instruction from said instruction store to check for a corresponding fast form instruction within said data store and, if said fast form instruction is present within said data store, then to execute said fast form instruction instead of said slow form instruction.

The invention recognises the above problems and provides the solution of using the instruction interpreter to check, upon encountering a slow form instruction whether or not a corresponding fast form instruction exists within the data store and, if present, to replace the slow form instruction with that fast form instruction. It has been found that the additional processing overhead associated with this check within the data store for a fast form of instruction is more than compensated for by the ability reliably to replace slow form instructions with fast form instructions with systems having a separate data store and instruction store.

It will be appreciated that the instruction interpreter could take many different forms. In particular, the instruction interpreter could be a hardware based instruction translator, a software based interpreter or a hybrid of the two.

It will be appreciated that whilst the separate data store and instruction store could take various different forms, the invention is particularly useful in embodiments having separate data caches and instructions caches.

The invention is particularly useful in embodiments in which an unresolved memory access is dynamically replaced by a resolved memory access. The unresolved memory access typically involves a symbolic reference to the data or instructions being sought whereas the resolved memory access will typically include a numeric reference to this information, the numeric reference being capable of direct use to return the required information and greatly increase speed.

The ability to properly replace slow form instructions with fast form instructions is particularly useful in embodiments in which the slow form instructions invoke additional processing procedures before completion, such as calls to further processing resources, which may even be on remote systems.

The ability to properly replace slow form instructions with fast form instructions is particularly useful when interpreting Java Virtual Machine instructions.

The instruction interpreter may in certain high performance embodiments of the invention where the advantage of properly replacing slow form with fast form instructions is particularly useful comprise an instruction translator for translating Java Virtual Machine instructions into native instructions of the processor core.

Viewed from another aspect the present invention provides a method of processing data using an apparatus having a processor core, a main memory operable to store instruction words and data words, a data store operable to store words from said main memory accessed by a data store port of said processor core, an instruction store operable to store words from said main memory accessed by an instruction store port of said processor core, and an instruction interpreter operable to read instruction words from said instruction store; said method comprising the steps of:

(i) modifying a slow form instruction within said instruction store to a fast form instruction of one or more possible fast form instructions and to write said fast form instruction to said data store, said slow form instruction and said fast form instruction having a common functionality when executed by said interpreter; and

(ii) upon reading a slow form instruction from said instruction store, checking for a corresponding fast form instruction within said data store and, if said fast form instruction is present within said data store, then executing said fast form instruction instead of said slow form instruction.

The above, and other objects, features and advantages of this invention will be apparent from the following detailed description of illustrative embodiments which is to be read in connection with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 schematically illustrates a Harvard type system within which the present invention may be utilised;

Figure 2 is a flow diagram illustrating the processing operations conducted in dealing with one type of slow form instruction;

Figure 3 illustrates a Java bytecode translator that may implement the invention; and

Figure 4 illustrates some ARM native instructions that may be used by a software interpreter to implement the invention.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Figure 1 illustrates a data processing system 2 including a processor core 4, an instruction cache 6, a data cache 8 and a main memory 10. The processor core 4 has an instruction access port that allows read only access to instructions within the instruction cache 6. Conversely, a data access port is provided that allows both read and write access to data words within the data cache 8. A unified external memory 10 is provided beyond the instruction cache 6 and the data cache 8.

In operation, instructions to be executed are read from the main memory 10 into the instruction cache 6 and then from the instruction cache 6 into the processor core 4 where they are executed. Data words required for the data processing operation specified by the instructions or generated by those instructions are read from or written to the data cache 8.

Figure 2 is a flow diagram illustrating the processing that may take place in the interpretation of a particular example slow form instruction. At step 12 an "invoke" Java bytecode instruction is read from the instruction cache 6. This "invoke" instruction is a slow form instruction that includes a symbolic reference to the process being invoked. It is known to provide interpreters that dynamically replace slow form instructions such as "invoke" with fast form instructions such as "invoke_quick". The fast form instruction "invoke_quick" includes a numeric reference to the processing code being called.

At step 14, the system makes a check at the instruction address of the "invoke" bytecode within the data cache 8 to see if an "invoke_quick" bytecode is already stored within the data

cache 8 at that address indicating that the slow form instruction has already been encountered and resolved into a fast form instruction in previous processing. If such a fast form instruction is present, then processing proceeds to step 16 at which the fast form "invoke_quick" instruction is executed instead of the slow form "invoke" instruction. If the fast form instruction is not present within the data cache 8, then processing proceeds to step 18 at which the slow form instruction is resolved into a fast form instruction. Step 20 writes the fast form instruction "invoke_quick" into the data cache 8 at the instruction address for the slow form instruction and then processing proceeds to step 16 at which the resolve fast form instruction "invoke_quick" is executed.

It will be appreciated that the above example is given in relation to one specific slow form instruction, namely "invoke". It will be appreciated that analogous processing operations may also be performed in respect of other slow form Java bytecode instructions such as:

anewarray;
checkcast;
getfield;
getstatic;
instanceof;
invokeinterface;
invokespecial;
invokestatic;
invokevirtual;
ldc;
ldc_w;
ldc2_w;
multianewarray;
new;
putfield; and
putstatic.

In each of these cases the respective fast form instructions to which the slow form instructions are resolved is given by:

anewarray_quick;
checkcast_quick;
getfield_quick;

getfield_quick_w;
getfield2_quick;
getstatic_quick;
getstatic2_quick;
instanceof_quick;
invokeinterface_quick;
invokenonvirtual_quick;
invokesuper_quick;
invokestatic_quick;
invokevirtual_quick;
invokevirtual_quick_w;
invokevirtualobject_quick;
ldc_quick;
ldc_w_quick;
ldc2_w_quick;
multianewarray_quick;
new_quick;
putfield_quick;
putfield_quick_w;
putfield2_quick;
putstatic_quick; and
putstatic2_quick.

It will be noted that there are more quick forms than slow forms. This is because a single slow form may map to different quick forms depending on the operands of the slow form, the size of operands being manipulated, the size of the operand index and other factors.

For example the slow operand getfield may map to one of getfield_quick getfield_quick_w or getfield2_quick as follows.

getfield -> getfield_quick

The opcode of this instruction was originall getfield, operating on a field determined dynamically to have an offset into the class instance data of 255 words or less and to have a width of one word.

getfield -> getfield_quick_w

The opcode of this instruction was originally getfield, operating on a field determined dynamically to have an offset into the class instance data of more than 255 words.

getfield -> getfield2_quick

The opcode of this instruction was originally getfield, operating on a field determined dynamically to have an offset into the class instance data of 255 words or less and to have a width of two words.

Here is a complete list of the mappings between slow and quick opcodes.

anewarray -> anewarray_quick
checkcast -> checkcast_quick
getfield -> getfield_quick
getfield -> getfield_quick_w
getfield -> getfield2_quick
getstatic -> getstatic_quick
getstatic -> getstatic2_quick
instanceof -> instanceof_quick
invokeinterface -> invokeinterface_quick
invokespecial -> invokenonvirtual_quick
invokespecial -> invokesuper_quick
invokestatic -> invokestatic_quick
invokevirtual -> invokevirtual_quick
invokevirtual -> invokevirtual_quick_w
invokevirtual -> invokevirtualobject_quick
ldc -> ldc_quick
ldc_w -> ldc_w_quick

ldc2_w -> ldc2_w_quick

multianewarray -> multianewarray_quick

new -> new_quick

putfield -> putfield_quick

putfield -> putfield_quick_w

putfield -> putfield2_quick

putstatic -> putstatic_quick

putstatic -> putstatic2_quick

A detailed description of this may be found in "The Java Virtual Machine Specification" (Edition 1) by Tim Lindholm and Frank Yellin published by Addison Wesley, ISBN 0-201-63452-X. Note that this information has been removed from Edition 2.

Figure 3 illustrates a hardware based instruction translator that may provide one embodiment of the invention. The hardware based instruction translator 22 includes hardware logic that recognises a particular slow form bytecode received. The instruction translator 22 may be present within the instruction processing pipeline of a processing system and accordingly will have access to the program counter address that is the bytecode address for the Java bytecode currently being translated. The bytecode address is represented as "BCAdd". Specific hardware 24 within the instruction translator 22 issues a lookup to the data cache 8 at the bytecode address BCAdd. If a Hit signal is returned, then this is accompanied by the replacement fast form instruction including its numeric reference and then this fast form instruction is used in place of the slow form instruction. In many cases, the fast form instruction is then passed from the instruction translator 22 to a complementary software interpreter as both the slow form instruction and the fast form instruction are too complex to be directly translated by the hardware translator 22. However, some fast form instructions are simple enough to be executed directly by the hardware translator 22, e.g. getfield_quick can be executed by hardware whereas the slow form is executed by software. Even though both of the slow form instruction and the fast form instruction are to be passed out to the software interpreter, the software interpreter is able to deal with the fast form instruction much more quickly than the slow form instruction since it already includes a resolve numeric address reference.

Figure 4 illustrates an example of some ARM processor instructions that may be used within a software interpreter to check whether or not a fast form instruction of an encountered slow form instruction is already present within the data cache 8. The first instruction loads into register R0 the contents of the data cache 8 corresponding to the bytecode address of the slow form instruction encountered. The second instruction compares the returned contents of the bytecode address from the data cache 8 with the bytecode for the fast form of the instruction. The third instruction branches to a routine that executes the returned fast form instruction if that has been found. If the branch is not taken, then the processing proceeds to resolve the slow form of the instruction into the fast form of the instruction after which the fast form of the instruction is executed.

Although illustrative embodiments of the invention have been described in detail herein with reference to the accompanying drawings, it is to be understood that the invention is not limited to those precise embodiments, and that various changes and modifications can be effected therein by one skilled in the art without departing from the scope and spirit of the invention as defined by the appended claims.